Corrigé **Exercises: Parallel merge and application to sort**

Jean-Louis Roch

I. Complexité de MERGE et algorithme séquentiel

1. Corrigé *On fixe, dans X, n positions arbitraires: il y a C_{n+m}^n possibilités. Une fois les positions fixées, on y place les n éléments (distincts) de A dans l'ordre. Et dans les espaces restants, tous les éléments étant distincts, il y a une seule possibilité pour placer les éléments de B dans l'ordre.*

2. Corrigé *A chaque opération de comparaison, on peut au mieux partitionner l'ensemble des sorties possibles en 2. Donc, un minorant de complexité est $\log_2 C_{n+m}^n$.*

3. Corrigé *On a $\log_2 n! \simeq n\log_2 n - n\log_2 e + \frac{1}{2}\log_2 n + O(1)$ d'où le minorant: $2n\log_2 2n - 2n\log_2 e - 2n\log_2 n + 2n\log_2 e - \log_2 n + \frac{1}{2}\log_2 n + O(1) \simeq 2n - frac12\log_2 n + O(1) \simeq 2n$.*

4.a. Corrigé *Dans la première boucle (la seule comportant des comparaisons), à chaque comparaison, on range 1 élement. Comme cette boucle range au plus $n+m-1$ éléments, il y a au plus $n+m-1$ comparaisons.*
De plus, pour l'instance $a_0 < \ldots < a_{n-2} < b_0 < \ldots < b_{m-1} < a_{n-1}$: les $n-1$ premiers éléments de A sont comparés à b_0 et les m éléments de B sont comparés à a_{n-1}. Le majorant $n+m-1$ est atteint.

4.b. Corrigé *Toutes les comparaisons de la première boucle for sont en dépendance séquentielle. Dans le cas général, on a $D(n,m) \geq \min(n,m)$. Dans l'instance précédente, toutes les comparaisons sont en dépendances. Donc $D(n,m) = n+m-1$ en pire cas.*

II. Un algorithme parallèle D&C pour MERGE

5.a. Corrigé *Comme A et B sont triés, A_1, A_2, B_1 et B_2 le sont aussi. Tous les éléments de A_1 et B_1 sont inférieurs à α, donc à tous les éléments de A_2 et B_2. On peut donc fusionner indépendamment les $n/2+j$ éléments de A_1 et B_1 aux $n/2+j$ premières positions de X; et les éléments de A_2 et B_2 aux $n+m-n/2-j$ dernières positions de X. On obtient bien ainsi un tableau trié.*
NB: si A ou B sont vides, il suffit de recopier -en parallèle- les éléments de l'autre tableau dans X.

5.b. Corrigé *On fait une recherche dichotomique dans B : elle nécessite $\lceil \log_2 m \rceil$ comparaisons.*

5.c Corrigé *La récurrence s'écrit directement; en pire cas, m est maximum ($n = m$) et le tableau B est toujours d'un seul côté d'où $D(n,n) \leq D(n/2,n) + \log n \leq D(n/2,n/2) + 2\log n - 1 = O(\log^2 n) = O(\log^2 n + m)$*

5.d. Corrigé $T_p < \frac{n+m}{p} + o(n+m) + O(\log^2 n + m)$.

III. Un algorithme très parallèle pour MERGE

6.a. Corrigé *Le nombre d'éléments de X inférieurs à a_i est i dans A et k dans B, donc $i+k$.*

6.b. Corrigé *En parallèle pour tout k: calcul du booléen $t_{i,k} := (b_{k-1} < a_i) \vee (b_k > a_i)$.*
Comme les éléments sont distincts, il est vrai pour un seul k.
Le calcul se fait en profondeur $O(1)$ avec au total m comparaisons.

6.c. Corrigé *On applique l'algorithme de la question 6.b à tous les éléments de A pour trouver leur position dans le tableau X en temps $O(1)$ avec $n.m$ comparaisons. D'où l'algorithme :*
- *En parallèle pour $i = 0..n-1$ doSEQ :*
  1. *En parallèle pour $k = 0..m-1$ calcul du booléen $t_{i,k} := (b_{k-1} < a_i) \vee (b_k > a_i)$.*
  2. *En parallèle pour $k = 0..m-1$: si $t_{i,k} == vrai$ alors $x_{i+k} := a_i$.*
- *On procède de même pour les éléments de B.*

On a ainsi fusionné les deux tableaux en temps $O(1)$ avec $2nm$ comparaisons.

IV. Un algorithme en cascade efficace pour MERGE

7.a. Corrigé *Chaque μ_i peut être interclassé comme en 6.a en temps 1 avec $O(\sqrt{m})$ opérations. Donc tous les μ_i sont calculés ainsi en temps 1 avec $O(\sqrt{n}.\sqrt{m}) = O(n+m)$ opérations.*

7.b. Corrigé *Une fois calculés le μ_i et μ_j on a une partition de A (resp. B) en $\sqrt{n}+\sqrt{m}$ blocs, définis pour A par les α_i et ν_j (resp. par les β_j et μ_i. On obtient ici en $O(1)$ une séquence de $\sqrt{n}+\sqrt{m}$ couples de blocs (un pour A, l'autre pour B); chaque couple contient un bloc de A avec moins de $\sqrt{n}$ à fusionner récursivement avec un bloc de B ayant moins de $\sqrt{m}$ éléments.*
On a $D(n) = D(\sqrt{n}) + O(1) = O(\log\log n)$ et $W_1(n) \leq \sqrt{n}.W_1(\sqrt{n}) + O(n) = O(n.\log\log n)$.

7.c. Corrigé *Il suffit de découper le tableau A (resp B) en $O(n/\log\log n)$ (resp. $O(m/\log\log m)$) blocs, et de prendre le premier élément de chaque bloc pour former le tableau α (resp. β).*
Ensuite on fusionne α et β en temps $O(\log\log n)$ avec $O(n)$ opérations.
Enfin, on a maintenant la liste des couples à fusionner; dans chaque couple il y a au plus $\log\log n$ éléments pour A et au plus $\log\log m$ éléments pour B. On fait une fusion séquentielle de ces blocs. L'algorithme est en $O(\log\log n)$ et chacune des 3 phases fait $O(n)$ opérations.

V. Application à un tri par fusion parallèle

8. Corrigé *On a $W_1(n) = 2W_1(N/2) + W_1^{(M)}(n)$ et $D(n) = D(N/2) + D^{(M)}(n)$. D'où par résolution de la récurrence dans les 3 cas:*
9.a. *$D(n) = O(n)$ et $W_1^(n) = O(n\log n)$;*
9.b. *$D(n) = O(\log^3 n)$ et $W_1^(n) = O(n\log n)$;*
9.c. *$D(n) = O(\log n.\log\log n)$ et $W_1^(n) = O(n\log n)$;*